Hochschule Mittweida
University of Applied Sciences
Fakultät Angewandte Computer- und Biowissenschaften

Eine Heuristik für das mehrkriterielle 3D-Bin-Packing-Problem

Bachelorarbeit

von

André Trobisch

Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme IVI

Betreuer:

Erster Prüfer Prof. Dr. rer. nat. Peter Tittmann
Zweiter Prüfer Dipl.-Math. (FH) Franziska Heinicke

Dresden, den 11. März 2016

Eidesstattliche Erklärung

Hiermit erkläre ich, André Trobisch, die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der von mir angegebenen Literatur verfasst zu haben.

Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Diese Arbeit hat in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen.

Dresden, den 11. März 2016

ANDRÉ TROBISCH

Zusammenfassung

Diese Arbeit behandelt die Entwicklung einer Packheuristik für ein spezielles 3-dimensionales Bin-Packing-Problem. Dazu werden Methoden verschiedener wissenschaftlicher Arbeiten miteinander kombiniert. Es wird ein Belademechanismus für allgemeine Bin-Packing-Probleme auf das spezielle Problem angepasst. Dieser versucht möglichst dichte Packungen zu produzieren. Weiter wird eine Heuristik, beruhend auf dem simulierten Abkühlen, zur Optimierung der Güte einer Lösung genutzt. Dies kann durch Veränderung der Packsequenzen erreicht werden, da ein deterministischer Belademechanismus verwendet wird.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Gegenstand dieser Arbeit ist das 3-dimsionale Bin-Packing-Problem (Behälterproblem) unter zusätzlichen Restriktionen. Dieses sei wie folgt definiert: Gegeben ist eine Menge quaderförmiger Boxen Γ und eine unendliche Menge ebenfalls quaderförmiger gleichgroßer Container (bins). Alle Boxen sind dabei unterscheidbar und besitzen die Attribute Höhe, Breite, Tiefe, Gruppe und Gewicht. Darüber hinaus sind bei einzelnen Boxen Rotationen um die x-, y- oder z-Achse erlaubt. Jeder Container besitzt die Eigenschaften Höhe H, Breite B und Tiefe T. Wir suchen nun nach einer zulässigen Anordnung aller Boxen in einer minimalen Anzahl von Containern. Als zulässig bezeichnen wir eine Anordnung, wenn folgende Eigenschaften erfüllt sind:

1. Boxen mit hohem Gewicht sind möglichst weit unten auf der Palette platziert.
2. Keine Box ragt aus dem beinhaltenden Container heraus.
3. Keine Box ragt in eine andere Box hinein.
4. Boxen gleicher Gruppe sind nacheinander abgreifbar, d.h. existieren zwei oder mehr Boxen einer Gruppe, dann ist jede dieser Boxen Nachbar mindestens einer Box derselben Gruppe, also auf, unter, neben, vor oder hinter dieser platziert.

Es handelt sich hierbei um eine Variante des Rucksackproblems (knapsack problem), eines Optimierungsproblem der Kombinatorik, welches NP-vollständig ist [6]. Daher soll in dieser Arbeit eine Heuristik als Lösungsverfahren zum Einsatz kommen. In der Praxis taucht das 3-dimsionale Bin-Packing-Problem

(3DBPP) überall dort auf, wo eine Menge quaderförmiger Gegenstände auf standardisierte Paletten, Container etc. verteilt werden soll. So zum Beispiel im Verpackungs- und Transportwesen. Wir werden verschiedene Konstruktions- und Verbesserungsheuristiken an die spezielle Problemstellung anpassen, kombinieren, erweitern und vergleichen. Im folgenden wird ein Überblick der Literatur zum 3DBPP gegeben. Abschnitt zwei und drei liefern die theoretischen Grundlagen und beschreiben die Heuristiken. Wir führen im vierten Abschnitt einige Benchmarkinstanzen für das 3DBPP ein und werten die Testergebnisse aus. Abschnitt fünf fasst die Arbeit zusammen.

1.2 Literatur

Einen einfachen Approximationsalgorithmus zur Lösung des allgemeinen Bin-Packing-Problems liefert Johnson in [4]. Dieser *First Fit Decreasing* genannte Algorithmus löst das Problem in polynomieller Zeit. Dabei werden zuerst alle Objekte nach absteigendem Gewicht sortiert und anschließend in den ersten Behälter gegeben, in dem noch Platz ist. Ist in keinem der geöffneten Behälter mehr Platz, wird ein neuer Behälter geöffnet. Auf eine Minimierung der Restkapazität zielt der *Best Fit Decreasing* Algorithmus ab. Ein Objekt wird dabei nicht in den ersten geöffneten Behälter eingefügt, sondern in jenen, in des es gerade noch passt. Beide Algorithmen haben bei einer Eingabegröße von n Objekten eine Laufzeit von $\mathcal{O}(n \cdot \log n)$. Die Laufzeit ergibt sich aus der Suche in einer sortierten Liste ($\mathcal{O}(\log n)$) für n Objekte. Beiden Algorithmen ist gemein, dass sie Eigenschaft 4 (Gruppenzusammenhang) nicht garantieren können. Der naive Algorithmus *Next Fit Decreasing* fügt die Objekte nacheinander in den letzten geöffneten Container, falls sie hineinpassen. Andernfalls wird der Container geschlossen, ein neuer geöffnet und das Objekt in den leeren Container eingefügt. Dieses Verfahren garantiert den Gruppenzusammenhang. Falkenauer und Delchambre verwenden in [3] einen genetischen Algorithmus um die optimale Lösung für ein Packungsproblem zu finden. Scheithauer [9] unterteilt den Container in verschieden hohe Schichten. Dabei definiert die Höhe der ersten Box, welche in eine Schicht eingefügt wird, die Höhe

der Schicht. Anschließen wird mit dem 2D-Next-Fit-Decreasing-Algorithmus die Schicht solange befüllt, wie Boxen hineinpassen. Passt eine Box nicht mehr in die aktuelle Schicht, so wird eine neue Schicht, mit der Höhe der einzufügenden Box, darüber begonnen und die Box eingefügt. Dieser Vorgang wird solange wiederholt bis alle Boxen eingefügt sind. Die Höhe des Containers ist dabei nicht beschränkt.

2 Grundlagen

2.1 Notation und Darstellung eines Packschemas

Container und Boxen sind stets im ersten Quadranten eines kartesischen Koordinatensystems wie in Abbildung 2.1 angeornet. Ihre Position wird durch die Koordinaten der hinteren linken Ecke ihrer Bodenfläche beschrieben. Die Breite B der Container verläuft entlang der x-Achse, die Tiefe T entlang der y-Achse und die Höhe H entlang der z-Achse. Die Koordinaten der Container sind immer $(0, 0, 0)$. Bei den Boxen verläft die Breite entlang oder parallel zur x-Achse, die Tiefe entlang oder parallel zur y-Achse und die Höhe entlang oder parallel zur z-Achse. Die Menge aller Boxen sei stets mit Γ bezeichnet. Die x-, y- und z-Koordinaten für eine Box notieren wir als x, y und z. Wir bezeichnen die Breite mit b, die Tiefe mit t, die Höhe mit h die Gruppe mit gr und das Gewicht mit g. Um festzustellen ob eine Box rotiert werden darf führen wir noch die zweiwertigen boolschen Variablen bt, ht und hb ein. Eine Box können wir nun als Tupel $(x, y, z, b, t, h, gr, g, bt, ht, hb)$ schreiben. Benötigen wir nur die x-Koordinate einer Box i so schreiben wir einfach i_x. Analog verfahren wir mit den restlichen Elementen eines Box-Tupels. Für Rotationen gelte:

$$i_{hb} = 0 \Rightarrow i \text{ darf um x-Achse rotiert werden} \quad \forall i \in \Gamma \tag{2.1}$$

$$i_{ht} = 0 \Rightarrow i \text{ darf um y-Achse rotiert werden} \quad \forall i \in \Gamma \tag{2.2}$$

$$i_{bt} = 0 \Rightarrow i \text{ darf um z-Achse rotiert werden} \quad \forall i \in \Gamma \tag{2.3}$$

Nehmen diese Variablen den Wert 1 an, sind die entsprechenden Rotationen nicht erlaubt.

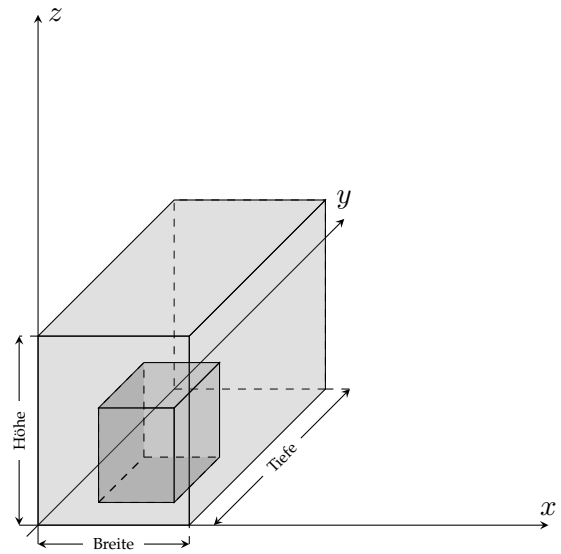

Abbildung 2.1: Ein Container mit zulässig platzierter Box

Um die Rotation einer Box beschreiben zu können, müssen wir noch ein paar Abbildungen definieren.

Definition 1.

Rotation um die x-Achse:

$$r_x : \Gamma \to \Gamma,$$
$$(x, y, z, b, t, h, gr, g, bt, ht, hb) \mapsto (x, y, z, h, t, b, gr, g, bt, ht, hb) \tag{2.4}$$

Rotation um die y-Achse:

$$r_y : \Gamma \to \Gamma,$$
$$(x, y, z, b, t, h, gr, g, bt, ht, hb) \mapsto (x, y, z, b, h, t, gr, g, bt, ht, hb) \tag{2.5}$$

Rotation um die z-Achse:

$$r_z : \Gamma \to \Gamma,$$
$$(x, y, z, b, t, h, gr, g, bt, ht, hb) \mapsto (x, y, z, t, b, h, gr, g, bt, ht, hb) \tag{2.6}$$

Es gelte:

Es existieren Kompositionen r_b, r_t, r_h der Rotationen sodass

$$r(i)_b \leq B, \quad r(i)_t \leq T, \quad r(i)_h \leq H, \quad \forall i \in \Gamma. \tag{2.7}$$

3 Die Heuristik

Die Heuristik setzt sich aus mehreren Modulen zusammen welche nacheinander durchlaufen werden. Abbildung 3.1 gibt einen Überblick über ihren Aufbau.

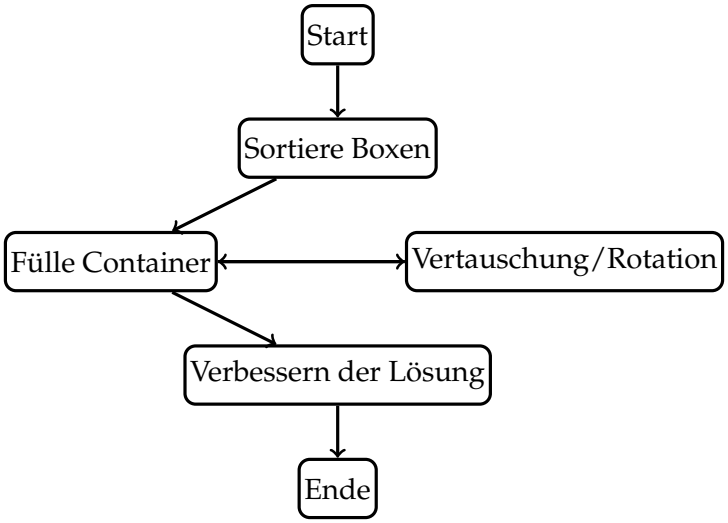

Abbildung 3.1: Schematischer Ablauf der Heuristik

3.1 Sortieren der Boxen

Im ersten Schritt werden die Boxen entsprechend ihrer Gruppe sortiert. Durch diese Vorsortierung sollen von den Konstruktionsheuristiken Packmuster erzeugt werden, welche das gruppenweise Entladen der Container ermöglichen.

3.2 2D-Bin-Packing

Zunächst wollen wir Verfahren für das 2-dimensionale Bin-Packing-Problem betrachten. Dieser Schritt ist notwendig, da die im folgenden Abschnitt betrachteten Methoden für das 3-dimensionale Bin-Packing-Problem darauf aufbauen. Dazu betrachten wir die Bodenfläche des Containers als Rechteck, welches es mit kleineren Rechtecken, den Grundflächen der Boxen, zu befüllen gilt sodass kein Rechteck über die Containergrundfläche hinausragt. Wir behalten die Bezeichnung Box und Container bei. Diese haben hierbei stets die Höhe 0.

3.2.1 Shelf-Next-Fit

Shelf-Algorithmen (Layer-Algorithmen) stellen einen recht einfachen Ansatz zur Lösung des Problems zur Verfügung. In unserem Fall werden die Boxen entlang der x-Achse angeordnet. Ist eine zulässige Platzierung auf diese Weise nicht mehr möglich wird zuerst geprüft, ob durch Rotation die Box doch noch eingefügt werden kann. Ist dies nicht möglich, so wird ein neuer Layer in y-Richtung angelegt und die Boxen erneut entlang der x-Achse platziert. Die Höhe eines Layers bestimmt die in diesem befindliche Box mit der größten Tiefe. Das Verfahren endet wenn alle Boxen platziert sind oder keine weitere mehr platziert werden kann. Abbildung 3.2 stellt den Ablauf des Algorithmus dar. Die Boxen sind entsprechend ihrer Gruppen eingefärbt. Eine Darstellung in Pseudocode zeigt Listing 3.1.

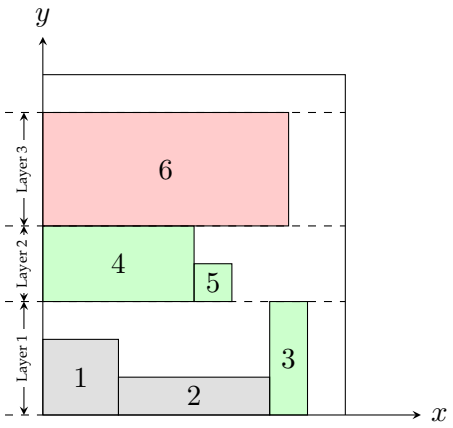

Abbildung 3.2: Shelf-Next-Fit-Algorithmus

Listing 3.1: Shelf-Next-Fit

```
INPUT: Container mit B und T, Boxen Γ

x = y = y_max = 0

for i in Γ:
  if x + i_b > B:
    if x + i_t ≤ B and y + i_b ≤ T:
      i_b, i_t = i_t, i_b
    else:
      x = 0
      y = y_max
      y_max = y + i_t
  if y + i_t > T:
    return
  else:
    i_x = x
    i_y = y
    x = x + i_b
    y_max = max(y_max, y + i_t)
```

Der Rechenaufwand für dieses Verfahren beträgt $\mathcal{O}(n)$ für $n = |\Gamma|$.

3.2.2 Guillotinen-Schnitt-Verfahren

Wie in Abbildung 3.2 leicht zu erkennen, kann beim Shelf-Next-Fit-Verfahren nach dem Schließen eines Layers der darin befindliche freie Raum nicht mehr für nachfolgende Boxen verwendet werden. Diesem Problem können wir mit dem Guillotinen-Schnitt-Verfahren [5] begegnen. Dieses basiert auf einem Verfahren bei welchem eine Box in der Ecke eines freien Rechteckbereichs des Containers platziert und anschließend der restliche Freiraum des Rechteckbereichs in zwei neue disjunkte Rechteckbereiche zerlegt wird. Abbildung 3.3 zeigt eine solche Zerlegung.

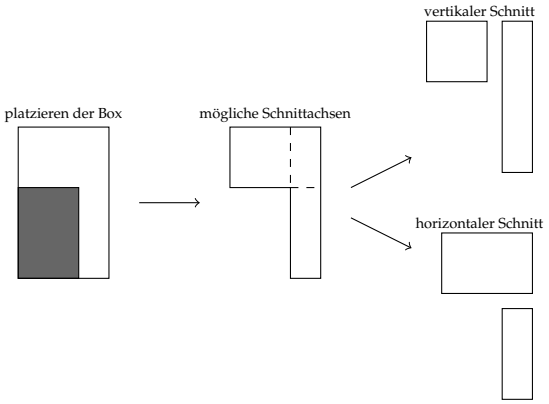

Abbildung 3.3: Guillotinen-Schnitt

Die freien Bereiche führen wir in einer Liste $\mathcal{F} = \{F_1, \ldots, F_n\}$. Dabei gilt $F_i \cup F_j = \varnothing \quad \forall i \neq j$. Die Vereinigung $\cup_{i=1}^{n} F_i$ gibt den gesamten ungenutzten Raum an. Das Verfahren beginnt mit einem einzelnen freien Rechteckbereich $\mathcal{F} = \{F_1 = (B, T)\}$. In jedem weiteren Schritt wird nun ein freier Bereich F_i gewählt, in welchem die nächste Box b platziert werden soll. Die Box b wird anschließend in der unteren linken Ecke von F_i positioniert und F_i entsprechend dem Guillotinen-Schnitt zerlegt. Dabei entstehen maximal zwei neue Rechteckbereiche $F^{'}$ und $F^{''}$. Diese werden der Liste $\mathcal{F}$ hinzugefügt und F_i wird daraus entfernt. Diese Prozedur wird solange wiederholt bis für eine Box, auch nach Rotation, kein passender Rechteckbereich mehr gefunden werden kann. In diesem Fall wird der aktuelle Container geschlossen

und ein neuer geöffnet. Das Verfahren endet wenn alle Boxen gepackt wurden. Bei diesem Vorgehen kann es passieren, dass Bedingung 4 (Boxen gleicher Gruppe sind nacheinander abgreifbar) verletzt wird. Abbildung 3.4 zeigt einen solchen Fall. Damit der Gruppenzusammenhang erhalten bleibt, kom-

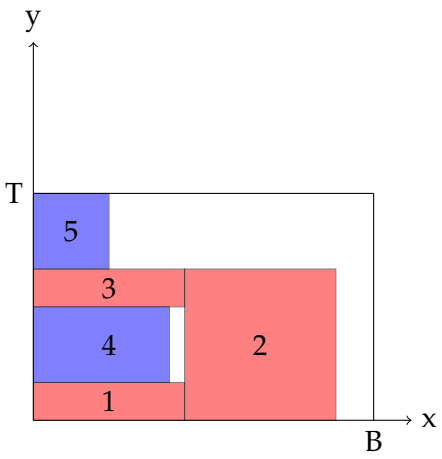

Abbildung 3.4: Verletzung des Gruppenzusammenhangs

binieren wir an dieser Stelle zwei der 2D-Bin-Packing Algorithmen. Zuerst werden alle Boxen einer Gruppe mit dem Guillotinen-Schnitt-Algorithmus in den Container geladen. Anschließend wird ein neuer Layer wie bei dem Shelf-Next-Fit-Algorithmus erzeugt. Seine y-Koordinate entspricht dabei der Oberkante der am weitesten in y-Richtung ragenden Box. Nun werden alle freien Rechteckbereiche welche sich unterhalb des neuen Layers befinden aus $\mathcal{F}$ gelöscht und der Guillotinen-Schnitt-Algorithmus mit der nächsten Gruppe fortgesetzt. Zur vollständigen Beschreibung des Verfahrens müssen nun noch weitere Regeln eingeführt werden. Julänki [5] stellt verschiedene Möglichkeiten vor, einen passenden Rechteckbereich F_i für eine Box b zu wählen.

- **Guillotine-Best-Area-Fit:** Von allen freien Bereichen in welche die Box platziert werden kann, wird jener mit dem minimalen Flächeninhalt gewählt.
- **Guillotine-Best-Short-Side-Fit:** Hierbei werden die Differenzen der Seitenlängen von b und F_i betrachtet. Es sei $F_{i,b}$ die Breite und $F_{i,t}$ die Tiefe

von F_i. Dann wird b in jenem F_i für welches $\min(F_{i,b} - b_b, F_{i,t} - b_t)$ den kleinsten Wert annimmt platziert. Es wird also die Länge der kürzeren resultierenden Seite minimiert.

- **Guillotine-Best-Long-Side-Fit:** Möchte man die längere resultierende Seite minimieren, so platziert man die Box b in jenem F_i für welches $\max(F_{i,b} - b_b, F_{i,t} - b_t)$ den kleinsten Wert annimmt.
- **Guillotine-Worst-Fit-Regeln:** Hierbei handelt es sich um Analogien zu den bereits beschriebenen Best-Fit-Techniken. So wird bei der **Guillotine-Worst-Area-Fit-Regel** F_i so gewählt, dass die dabei entstehenden neuen Rechteckbereiche möglichst großen Flächeninhalt haben. Dabei gilt für alle Guillotinen-Methoden eine Sonderregel: Gilt $F_{i,b} = b_b$ und $F_{i,t} = b_t$ für ein i, so wird b in F_i platziert. So können wir auch die **Guillotine-Worst-Short-Side-Fit-Regel**, wobei die Länge der verbleibenden kürzeren Seite maxiert wird, und die **Guillotine-Worst-Long-Side-Fit-Regel**, bei welcher wir die Länge der verbleibenden längeren Seite maximieren, definieren. Die Idee hinter den Worst-Fit-Regeln ist dabei solange wie möglich große Freiflächen zu erhalten und so kleine Rechteckbereiche, in welche keine Box mehr platziert werden kann, zu vermeiden.

Satz 1. *Die bisher betrachteten Guillotinen-Schnitt-Verfahren können so implementiert werden, dass für diese eine Laufzeit von* $\mathcal{O}(n^2), \quad n = |\Gamma|$ *erreicht wird.*

Beweis. Diese Algorithmen unterscheiden sich lediglich in der Art und Weise wie zwei Elemente miteinander verglichen werden. Dies geschieht in allen Fällen in kostanter Zeit. Der Wert von $|\mathcal{F}|$ hat eine Wachstumsrate von $\mathcal{O}(n)$ da in jedem Iterationsschritt maximal ein Rechteckbereich zu $\mathcal{F}$ hinzugefügt wird. Für jede Box müssen alle freien Rechteckbereiche untersucht werden. Dies führt zu einer Laufzeit von $\mathcal{O}(n^2)$. □

Rectangle-Merge-Improvement

Ein weiteres Problem ist der Umstand, dass Boxen nicht auf mehreren freien Bereichen platziert werden können. Seien zum Beispiel b eine Box und

$F_1, \ldots, F_k \in \mathcal{F}$ freie Rechteckbereiche. Angenommen b kann in $F_1 \cup \ldots \cup F_k$ platziert werden, nicht aber in $F_i, \quad i = 1 \ldots, k$. Dann findet der Algorithmus keinen Raum zum platzieren der Box obwohl dieser vorhanden ist. Es gilt also die Fragmentierung des Freiraums zu reduzieren. Julänki schlägt vor in jedem Iterationsschritt für alle Paare benachbarter Freiflächen F_i, F_j zu prüfen, ob $F_i \cup F_j$ exakt durch eine einzelne größere rechteckige Fläche dargestellt werden kann. Ist dies der Fall so werden F_i und F_j aus $\mathcal{F}$ entfernt und $F_i \cup F_j$ hinzugefügt. Dieses Vorgehen führt dann zu einer geringeren Fragmentierung der Gesamtfreifläche.

Satz 2. *Das Guillotinen-Schnitt-Rectangle-Merge-Improvement-Verfahren kann so implementiert werden, dass für dieses eine Laufzeit von $\mathcal{O}(n^3), \quad n = |\Gamma|$ erreicht wird.*

Beweis. Nach jedem Platzieren müssen alle Paare $F_i, F_j \in \mathcal{F}$ untersucht werden. Es gibt $\Theta(n^2)$ solcher Paare und dies führt zu einer Gesamtlaufzeit von $\mathcal{O}(n^3)$. □

Wir wollen nun noch die möglichen Regeln für den Guillotinen-Schnitt betrachten. Auch hier schlägt Julänki sechs verschiedene Möglichkeiten vor. Diese Betrachtungen sind wichtig da die Schnittachse die Größe der resultierenden Freiflächen bestimmt und eine Box möglicherweise nicht über jener Schnittachse platziert werden kann. Im Folgenden sei eine Box b platziert auf einer Freifläche F_i mit Breite $F_{i,b}$ und Tiefe $F_{i,t}$.

- **Shorter/Longer-Axis-Split-Regel:** Unabhängig von den Abmessungen von b wird hier die Freifläche horizontal geteilt falls $F_{i,b} < F_{i,t}$ und vertikal falls $F_{i,b} \geq F_{i,t}$. Für die Longer-Axis-Split-Regel gelten die umgekehrten Relationszeichen.
- **Shorter/Longer-Leftover-Axis-Split-Regel:** Es werden hier die Längen der nach dem Platzieren von b verbliebenen Schnittachsen betrachtet. So wird die Freifläche horizontal geteilt falls gilt $F_{i,b} - b_b < F_{i,t} - b_t$, andernfalls vertikal. Bei der Longer-Leftover-Axis-Split-Regel gelten wiederum die umgekehrten Relationszeichen.

- **Max/Min-Area-Split-Regel:** Zunächst zerlegen wir F_i, wie in Abbildung 3.5 zu sehen, in die Freiflächen A_1, A_2, A_3 sowie in einen Bereich in welchem b platziert ist. Nun wird F_i aus $\mathcal{F}$ entfernt und bei Anwendung

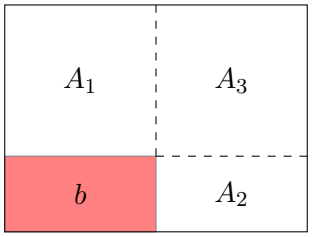

Abbildung 3.5: Max/Min-Area-Split-Regel

der Max-Area-Split-Regel $A_1 \cup A_3$ und A_2 hinzugefügt sofern A_1 kleiner als A_2 ist. Andernfalls wird A_2 mit A_3 vereinigt und die Freiflächen zu $\mathcal{F}$ hinzugefügt. Bei der Min-Area-Split-Regel wird A_3 hingegen immer mit der größeren verbliebenen Freifläche vereinigt.

Alle Regeln für den Guillotinen-Schnitt lassen sich in kostanter Zeit berechnen und haben somit keinen Einfluss auf die Komplexität des Gesamtalgorithmus. Eine Pseudocodedarstellung für das Guillotinen-Schnitt-Verfahren zeigt Listing 3.2.

Listing 3.2: Guillotinen-Schnitt-Verfahren

```
1   INPUT: Container mit B und T, Boxen Γ
2
3   F = {(B,T)}
4   G = {G_1,...,G_n} // Menge aller Gruppen
5   // ∪_{i=1}^n G_i = Γ
6   // ∀a,b ∈ G_i : a_gr = b_gr,   i = 1,...,n
7   y_max = 0
8
9   for G ∈ G:
10    F = {F ∈ F|F liegt nicht unterhalb y_max}
11
12    for b ∈ G:
13      finde F ∈ F in welches b gepackt wird
14
15      if es existiert kein solches F:
16        oeffne einen neuen Container
17        F = {F = (B,T)}
18
19      platziere b in der linken unteren Ecke von F
20      y_max = max(y_max, b_y + b_t)
21      zerlege F in F_1, F_2
22      F = F ∪ {F_1, F_2} \ F
```

3.2.3 Maximal-Rechteck-Verfahren

Bei dieser Methode handelt es sich um eine Erweiterung des Guillotinen-Schnitt-Verfahrens. Dabei wollen wir die Restriktionen, welche durch die Schnittachsen hervorgerufen werden, beseitigen. Auch hier führen wir wieder eine Liste $\mathcal{F} = \{F_1, \ldots, F_n\}$ mit freien Rechteckbereichen. Diese müssen dabei nicht disjunkt sein. Anstsatt uns nun für eine Schnittachse zu entscheiden, werden nun beide Schnitte ausgeführt. Abbildung 3.6 zeigt das Vorgehen. Wird also eine Box b auf einem Rechteckbereich F_i platziert, werden die Freiflächen F_1 und F_2 mit maximaler Länge in beide Richtungen berechnet. Diese

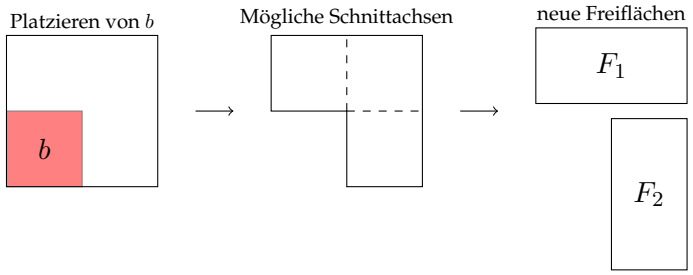

Abbildung 3.6: Schnittregel des Maximal-Rechteck-Verfahrens

werden zu $\mathcal{F}$ hinzugefügt und F_i daraus entfernt. Dieses Vorgehen führt zu der folgenden Eigenschaft von $\mathcal{F}$:

Satz 3. *Sei $\mathcal{F} = \{F_1, \ldots, F_n\}$ die Menge aller freien maximalen Rechteckbereiche. Falls eine Box b auf $\cup_{i=1}^{n} F_i$ zulässig platziert werden kann, so existiert auch ein $F_i \in \mathcal{F}$ auf welchem b zulässig platziert werden kann.*

Beweis. Bei der Suche nach einer passenden Freifläche für eine Box können alle $F_i \in \mathcal{F}$ in einem Container untersucht werden und falls eine zulässige Platzierung existiert, so wird diese auch gefunden. □

Da die Elemente von $\mathcal{F}$ nun nicht mehr disjunkt sind, müssen wir nun nach dem Platzieren jeder Box b in eine Fläche F_i alle anderen $F_j \in \mathcal{F}$ überprüfen und falls $b \cap F_j \neq \emptyset$ gilt, diese mit b schneiden und auf diese Weise neue Freiflächen erzeugen. Bei diesem Schritt können doppelte oder nicht-maximale Freiflächen in $\mathcal{F}$ entstehen. Daher ist es notwendig für jedes F_i zu prüfen ob ein $F_j \in \mathcal{F}, i \neq j$ existiert, sodass $F_i \subseteq F_j$. Ist das der Fall, so wird F_i aus $\mathcal{F}$ entfernt.

Da auch bei dem Maximal-Rechteck-Verfahren die Möglichkeit besteht, dass Boxen unzulässig hinsichtlich des Gruppenzusammenhangs platziert werden, muss diese Methode in gleicher Weise wie das Guillotine-Schnitt-Verfahren um die Layereigenschaft erweitert werden. Den entsprechenden Pseudocode zeigt Listing 3.3.

Listing 3.3: Maximal-Rechteck-Verfahren

```
1  INPUT: Container mit B und T, Boxen Γ
2
3  F = {(B,T)}
4  G = {G_1,...,G_n} // Menge aller Gruppen
5  // ∪_{i=1}^n G_i = Γ
6  // ∀a,b ∈ G_i : a_gr = b_gr,   i = 1,...,n
7  y_max = 0
8
9  for G ∈ G:
10   F = {F ∈ F | F liegt nicht unterhalb y_max}
11
12   for b ∈ G:
13     finde F ∈ F in welches b gepackt wird
14
15     if es existiert kein solches F:
16       oeffne einen neuen Container
17       F = {F = (B,T)}
18
19     platziere b in der linken unteren Ecke von F
20     y_max = max(y_max, b_y + b_t)
21     zerlege F in F_1, F_2 //maximale Rechtecke
22     F = F ∪ {F_1, F_2} \ F
23
24     for F ∈ F:
25       berechne F \ b
26       //ergibt maximal vier neue Bereiche F_1,...,F_4
27       F = F ∪ {F_1,...,F_4} \ F
28
29     for F_i, F_j ∈ F: //jedes geordnete Paar
30       if F_i in F_j enthalten:
31         F = F \ F_i
```

Satz 4. *Das Maximal-Rechteck-Verfahren kann so implementiert werden, dass es eine Laufzeit von $\mathcal{O}(|\mathcal{F}|^2|\Gamma|)$ erreicht.*

Beweis. Nachdem jede Box platziert und mit allen Elementen aus $\mathcal{F}$ geschnitten wurde, wird über alle Paare von Freiflächen iteriert um doppelte nicht maximale Flächen zu entfernen. Dieser zeitaufwändige Schritt führt zu einer Laufzeit von $\mathcal{O}(|\mathcal{F}|^2|\Gamma|)$. □

Alle Regeln zur Wahl des Rechteckbereiches in welches eine Box platziert werden soll, welche auch bei dem Guillotinen-Schnitt-Verfahren zum Einsatz kommen, können hier unverändert übernommen werden. Dies liefert uns folgende Regeln:

- Maximal-Rectangle-Best-Area-Fit-Regel
- Maximal-Rectangle-Best-Short-Side-Fit-Regel
- Maximal-Rectangle-Best-Long-Side-Fit-Regel
- Maximal-Rectangle-Worst-Area-Fit-Regel
- Maximal-Rectangle-Worst-Short-Side-Fit-Regel
- Maximal-Rectangle-Worst-Long-Side-Fit-Regel

In [5] findet sich hierzu noch eine weiterer Vorschlag. Bei diesem Maximal-Rectangles-Bottom-Left- oder Tetris-Algorithmus genannten Verfahren werden die Boxen so platziert, dass die y-Koordinate der Oberkante einer Box minimal wird. Falls dabei mehrere Möglichkeiten auftauchen, wird jene Freifläche mit der kleinsten x-Koordinate gewählt. Abbildung 3.7 zeigt ein mögliches Packmuster.

3.3 3D-Bin-Packing

In diesem Abschnitt wollen wir die soeben vorgestellten Verfahren auf den 3-dimensionalen Fall erweitern.

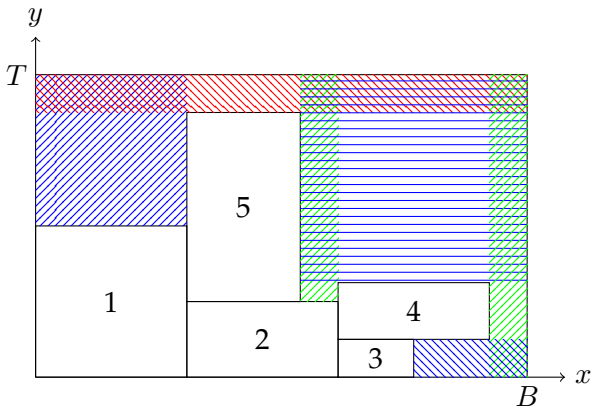

Abbildung 3.7: Ein durch den Tetris-Algorithmus erzeugtes Packmuster

3.3.1 Shelf-Next-Fit (3D)

Wir nutzen das Modell von Scheithauer [9]. In diesem definiert die erste platzierte Box die Höhe des ersten Layers parellel zur Bodenfläche. Danach wird mit einem 2D-Bin-Packing-Algorithmus die Bodenfläche des Containers mit den rechteckigen Bodenflächen der Boxen solange nacheinander befüllt, bis entweder alle Boxen platziert sind oder keine weitere in den Container passt. Kann eine Box nicht mehr in einen Layer gepackt werden, so wird dieser geschlossen und ein neuer Layer mit der Höhe der ersten zu platzierenden Box geöffnet.

Auch hier zwingt uns die Gruppenrestriktion zu einer weiteren Anpassung. Um die Höhe eines Layers zu bestimmen, nutzt Scheithauer die Höhe der ersten darin platzierten Box. Das ist möglich da in seinem Modell alle Boxen der Höhe nach in absteigender Reihenfolge sortiert sind. Eine solche Sortierung ist in unserem Fall nicht möglich. Daher wird die z-Koordinate eines Layers durch die höchste darin befindliche Box definiert. Dies kann einfach durch das Mitführen einer Variablen, welche nach dem Platzieren einer Box aktualisiert wird, erreicht werden. Das Mitführen der Variablen und die zusätzliche Dimension führen nur zu einer konstanten Anzahl zusätzlicher Rechenoperationen und haben somit keinen Einfluss auf die Komplexität des Gesamtalgorithmus gegenüber dem 2-dimensionalen Verfahren.

3.3.2 Guillotinen-Schnitt-Verfahren (3D)

An Stelle freier Rechteckbereiche führen wir nun eine Liste $\mathcal{F} = \{F_1, \ldots, F_n\}$ mit freien Quaderbereichen. Diese seien wieder paarweise disjunkt und die Vereinigung aller dieser Quaderbereiche liefert uns das Gesamtvolumen des nicht genutzten Raumes in einem Container. Zu Beginn des Verfahrens befindet sich in $\mathcal{F}$ genau ein solcher Quader F_i mit Breite $F_{i,b} = B$, Tiefe $F_{i,t} = T$ und Höhe $F_{i,h} = H$. Analog zu der 2-dimensionalen Variante wird in jedem Iterationsschritt eine Box in der linken hinteren Ecke der Bodenfläche eines freien Quaders platziert und und anschließend die resultierenden Freiräume berechnet. Aufgrund der höheren Dimension entstehen hier zusätzliche Auswahlmöglichkeiten zur Positionierung der Schnittflächen. Diese sind in Abbildung 3.8 dargestellt. Man erkennt leicht dass bei diesem Vorgang ma-

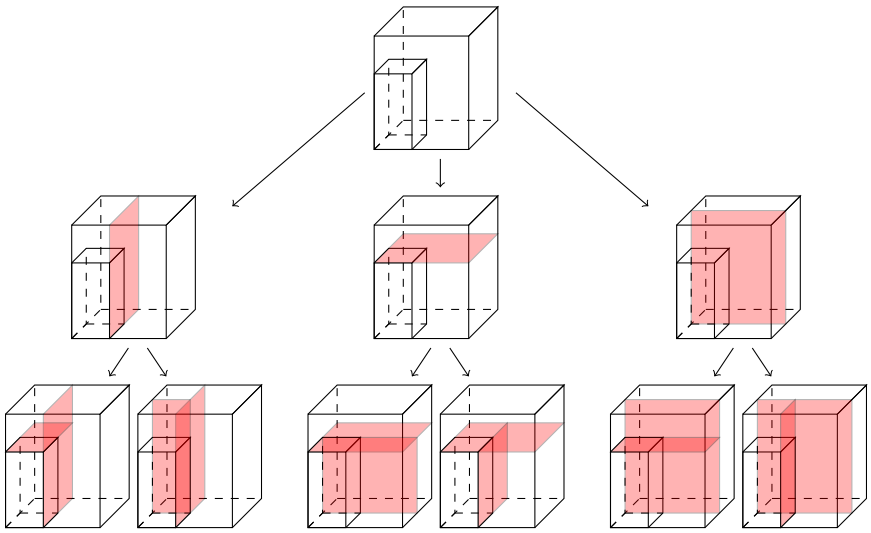

Abbildung 3.8: Guillotinen-Schnitt: Mögliche Schnittflächen

ximal drei neue freie Quader entstehen, deren Abmessungen wiederum sind abhängig von der Wahl der Schnittflächen. Nun können wir wieder die Regeln für den Guillotinen-Schnitt definieren. Bereits aus dem 2-dimensionalen Guillotinen-Schnitt-Verfahren konnten wir 36 verschiedene Algorithmen ableiten (sechs Methoden zur Auswahl eines freien Rechteckbereichs kombinier-

bar mit sechs Methoden zur Durchführung des Schnittes).
Bei der 3-dimensionalen Variante sind nun zwei Schnitte hintereinander auszuführen. Zunächst lassen sich alle bereits zur Auswahl der Schnittachsen bekannten Regeln übertragen. Dazu wendet man diese einfach auf die in Abbildung 3.8 rot gekennzeichneten Flächen an. Hinzu kommen nun noch die Möglichkeiten anhand der Kantenlängen der resultierende Quader sowie deren Volumina über ein Schnitt zu entscheiden. Ebenso wachsen die Möglichkeiten bei der Wahl eines freien Quaders in welchen eine Box platziert werden soll. Zusätzlich zu den Verhältnissen der Kantenlängen und Seitenflächen der entstehenden Quader, vor bzw. nach Platzieren einer Box, können nun noch die resultierenden Volumina miteinander verglichen werden.

Auch bei diesem Verfahren müssen wir den Gruppenzusammenhang der Boxen im Blick behalten. Dazu führen wir wieder die gruppenweise Layer-Restriktion ein. Diese jedoch nur in z- und nicht auch in y-Richtung, da sonnst einzelne Boxen isoliert werden könnten. In Listing 3.4 ist der Ablauf des Algorithmus dargestellt.

Listing 3.4: Guillotinen-Schnitt-Verfahren, 3D-Variante

```
1   INPUT: Container mit B, T, und H; Boxen Γ
2
3   F = {(B,T,H)}
4   G = {G_1,...,G_n} // Menge aller Gruppen
5   // ∪_{i=1}^n G_i = Γ
6   // ∀a,b ∈ G_i : a_gr = b_gr,   i = 1,...,n
7   z_max = 0
8
9   for G ∈ G:
10    F = {F ∈ F| Oberkante von F liegt ueber oder auf z_max}
11
12    for b ∈ G:
13      finde F ∈ F in welches b gepackt wird
14
15      if es existiert kein solches F:
16        oeffne einen neuen Container
17        F = {F = (B,T)}
18        z_max = 0
19
20      platziere b in der linken unteren Ecke von F
21      z_max = max(z_max, b_z + b_h)
22      zerlege F in F_1, F_2, F_3
23      F = F ∪ {F_1, F_2, F_3} \ F
```

Die Komplexität des Gesamtalgorithmus ist auch unter Berücksichtigung von Rotationen gleich seiner 2-dimensionalen Variante.

3.3.3 Maximal-Quader-Verfahren

Nun erweitern wir das Maximal-Rechteck-Verfahren auf den 3-dimensionalen Fall. Auch hier sollen die, durch die Schnittflächen im 3D-Guillotinen-Schnitt-Verfahren hervorgerufenen Restriktionen beseitigt werden.
Es sei $\mathcal{F} = \{F_1, \ldots, F_n\}$ also eine Liste freier Quader welche zu Beginn genau ein Element mit den Abmessungen des Containers enthält. Um nun ei-

ne Box b zu platzieren stehen uns die gleichen Methoden wie auch beim 3-dimensionalen Guillotinen-Schnitt-Verfahren zur Verfügung. Ist b erst einem $F \in \mathcal{F}$ zugeordnet, so müssen für F nur die ersten drei Schnittmöglichkeiten des Guillotinen-Schnitt-Verfahrens betrachtet werden um die resultierenden freien Quader zu bestimmen. In diesem Schritt entstehen maximal drei neue Quader F_1, F_2, F_3. Abbildung 3.9 illustriert den Vorgang. Nach dem Schnitt

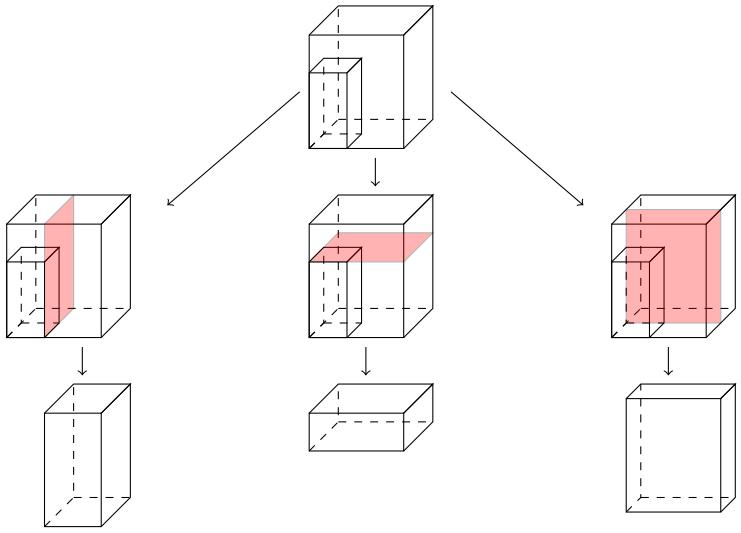

Abbildung 3.9: Maximal-Quader: Mögliche Schnittflächen

werden F_1, F_2 und F_3 zu $\mathcal{F}$ hinzugefügt und F daraus entfernt. Auch hier gilt falls $b \subseteq \cup_{i=1}^{n} F_i$, so existiert auch ein $F_i \in \mathcal{F}$ in mit $b \subseteq F_i$. Alle Elemente aus $\mathcal{F}$ werden nun mit b geschnitten, sofern diese nicht disjunkt sind, und anschließend alle nicht maximalen und doppelten freien Quader auf $\mathcal{F}$ entfernt. Auch hier gilt die gruppenweise Layer-Restriktion wie wir sie bereits bei dem 3-dimensionalen Guillotinen-Schnitt-Verfahren gesehen haben. Listing x 3.5 den Ablauf des Verfahrens in Pseudocode dar.

Listing 3.5: Maximal-Quader-Verfahren, 3D-Variante

```
1  INPUT: Container mit B, T und H; Boxen Γ
2
3  F = {(B,T,H)}
4  G = {G_1,...,G_n} // Menge aller Gruppen
5  // ∪_{i=1}^n G_i = Γ
6  // ∀a,b ∈ G_i : a_gr = b_gr,   i = 1,...,n
7  z_max = 0
8
9  for G ∈ G:
10   F = {F ∈ F | F liegt nicht unterhalb z_max}
11
12   for b ∈ G:
13     finde F ∈ F in welches b gepackt wird
14
15     if es existiert kein solches F:
16       oeffne einen neuen Container
17       F = {F = (B,T,H)}
18
19     platziere b in der linken hinteren Ecke auf dem Boden
           von F
20     z_max = max(z_max, b_z + b_h)
21     zerlege F in F_1,F_2,F_3 //maximale Quader
22     F = F ∪ {F_1,F_2,F_3} \ F
23
24     for F ∈ F:
25       if b ∩ F ≠ ∅:
26         //ergibt maximal sechs neue Bereiche F_1,...,F_6
27         F = F ∪ {F_1,...,F_6} \ F
28
29     for F_i,F_j ∈ F: //jedes geordnete Paar
30       if F_i ⊆ F_j:
31         F = F \ F_i
```

Die Komplexität des Gesamtalgorithmus kann mit $\mathcal{O}(|\mathcal{F}|^2|\Gamma|)$. angegeben werden und entspricht somit der 2-dimensionalen Form des Verfahrens.

In Listing 3.5 sehen wir, dass falls b mit einem freien Quader F geschnitten wird, nun bis zu sechs neue Quader entstehen können. Das kommt daher da b nicht immer in einer Ecke von F liegen muss. Abbildung 3.10 zeigt einen solchen Fall.

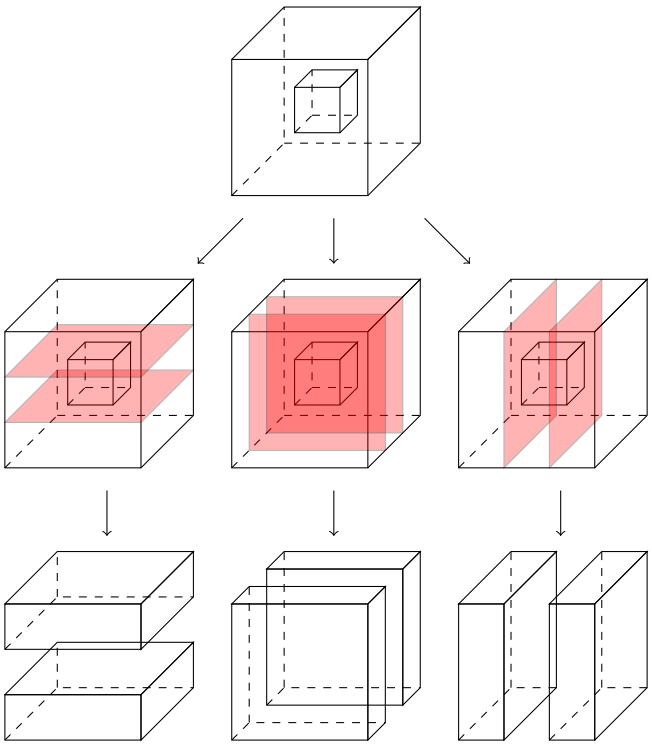

Abbildung 3.10: Erzeugen der Schnittquader

3.4 Verbessern der Lösung

Kostenfunktion

Um die Güte einer Lösung zu bestimmen müssen wir noch eine Kostenfunktion f definieren. Da die Anzahl verfügbarer Container unbegrenzt ist und den Boxen bzw. den Gruppen keine Kosten zugeordnet sind, können wir f einfach mit Hilfe genutzten Höhe einer Packsequenz herleiten. Um die Anzahl der

Container möglichst klein zu halten sind dichte Packsequenzen zu bevorzugen. Betrachtet man nun einen einzelnen Container, dann folgt daraus, dass die Oberkante der am höchsten platzierten Box in diesem Container niedriger ist als bei weniger dichten Packsequenzen. Es sei eine Packsequenz bestehend aus der Menge aller enthaltenen Boxen Γ, den verwendeten Containern $C_1, \ldots, C_n$ sowie deren Abmessungen H, B und T gegeben. Dann können wir die Kostenfunktion wie folgt definieren:

$$f = \sum_{i=1}^{n} \max\{b_z + b_h | b \text{ ist in } F_i \text{ platziert}\}. \tag{3.1}$$

Diese Funktion gilt es nun zu minimieren.

3.4.1 Simuliertes Abkühlen

Da das Bin-Packing-Problem NP-vollständig ist, nutzen wir ein heuristisches Optimierungsverfahren zum Auffinden einer approximativen Lösung. Dabei wird der Abkühlprozess eines glühenden Werkstoffes nachgebildet nachgebildet. Die Temperatur entspricht hierbei der Wahrscheinlichkeit, mit welcher sich ein Zwischenergebnis der Optimierung verschlechtern darf um dennoch akzeptiert zu werden. Dieses vorgehen hat den Vorteil gegenüber der lokalen Suche, dass ein lokales Optimum wieder verlassen und ein besseres Ergebnis gefunden werden kann. Der Zustand einer Instanz sei hierbei einfach durch eine Liste von Boxen gegeben. Da die Packalgorithmen deterministisch ablaufen, wird durch jeden Zustand wiederum eine Lösung repräsentiert.

Nachbarschaftsbeziehung

Für das Simulierte Abkühlen werden Nachbarn der aktuellen Packsequenz betrachtet. Diese entstehen dadurch, dass einzelne Boxen in der Ausgangsliste vertauscht und/oder rotiert werden. Dabei ist darauf zu achten, dass die Vorsortierung nach Gruppen erhalten bleibt. Auch können ganze Gruppen innerhalb einer Liste vertauscht werden.

Um das Abkühlen zu simulieren orientieren wir uns an der Arbeit von Väth [11] welche wiederum auf einen Beitrag von Ceschia und Schaerf [1] aufbaut. Dabei wird ein Startwert T_0 mittels eines Faktors $0 < \tau < 1$ abgesenkt, bis der Grenzwert T_{min} erreicht ist. Dieser stellt das Abbruchkriterium dar. In jeden Schritt wird nun eine vorgegebene Anzahl von σ_N Iterationen durchgeführt, bei welcher jeweils ein Nachbar ermittelt wird. Die Wahrscheinlichkeit p mit welcher wir einen solchen Nachbarn mit einer schlechteren Lösung akzeptieren, nimmt exponentiell mit der Zeit ab. Konkret ist also $p = e^{-c_N/T}$, wobei c_N die Kosten des Nachbarn wiedergeben. Das simulierte Abkühlen ist in Listing 3.6 dargestellt.

Listing 3.6: Simuliertes Abkühlen

```
1   c_min = f(startloesung)
2   besteLoesung = startloesung
3   T = T_0
4   while T > T_min:
5     for i = 1,...,σ_N:
6       nachbar = erzeugeNachbar(besteLoesung)
7       loesung = berechneLoesung(nachbar)
8       c_N = f(loesung)
9       if c_N < c_min or get(p): /* get(p) gibt mit
            Wahrscheinlichkeit p true zurueck, andernfalls
            false */
10        besteLoesung = loesung
11        c_min = c_N
12      T = T · τ
13  return besteLoesung
```

3.4.2 Austauschen von Boxen

In der Praxis kann es häufig vorkommen, dass einige Boxen, auch unter Berücksichtigung möglicher Rotationen, gleiche Abmessungen haben. Da die Boxen aber individuell gefüllt werden können diese ein unterschiedliches Ge-

wicht aufweisen. Aus diesem Grund werden noch einmal alle Boxen miteinander verglichen. Haben zwei Boxen a, b gleiche Abmessungen und es gilt $a_g > b_g \wedge a_z > b_z \wedge a_{gr} = b_{gr}$, so werden die Boxen vertauscht. Die ist in Listing 3.7 dargestellt.

Listing 3.7: Tauschen ähnlicher Boxen

```
1   INPUT: Boxen Γ
2
3   G = {G_1, ..., G_n} // Menge aller Gruppen
4   // ∪_{i=1}^{n} G_i = Γ
5   // ∀a, b ∈ G_i : a_gr = b_gr,   i = 1, ..., n
6
7   for G ∈ G:
8     for a, b ∈ G: // alle Paare
9       if a_g > b_g ∧ a_z > b_z:
10        tausche a und b
```

4 Vergleich

In diesem Abschnitt werden nun die Implementierungen vorgestellt und die Güte der Rechenergebnisse sowie die Laufzeit der Algorithmen miteinander verglichen.

4.1 Daten

Um einen Testfall zu generieren wurde für jede Probleminstanz zunächst eine feste Anzahl an Containern festgelegt und diese entlang ihrer z-Achse in verschiedene Bereiche zufälliger Höhe zerteilt. Diese Bereiche wiederum wurden anschließend mehrfach nach dem Zufallsprinzip parallel zur x-y-, x-z- sowie y-z-Achse zerteilt und die so entstandenen Schnittbereiche zu einer Gruppe von Boxen zusammengefasst. Anschließend wurde einzelnen Boxen noch die Möglichkeit zur Rotation gegeben und eine Teilmenge von diesen rotiert. Diese Art der Probleminstanzerzeugung bringt den Vorteil mit sich, dass sich die idealen Kosten einfach aus dem Produkt der Containeranzahl und der Höhe des Conatiners berechnen lassen. Somit stehen uns auch gleich die Referenzwerte für eine ideale Packung zur Verfügung.

4.2 Effizienzvergleich

Es werden nun Shelf-Next-Fit-, Guillotinen-Schnitt- und Maximal-Quader-Algorithmus hinsichtlich ihrer Effizienz und Güte miteinander verglichen. Da die Probleminstanzen große praxisnähe aufweisen sollten, sind Rotationen einzelner Boxen um die x- oder y-Achse nur bei sehr wenigen Boxen mög-

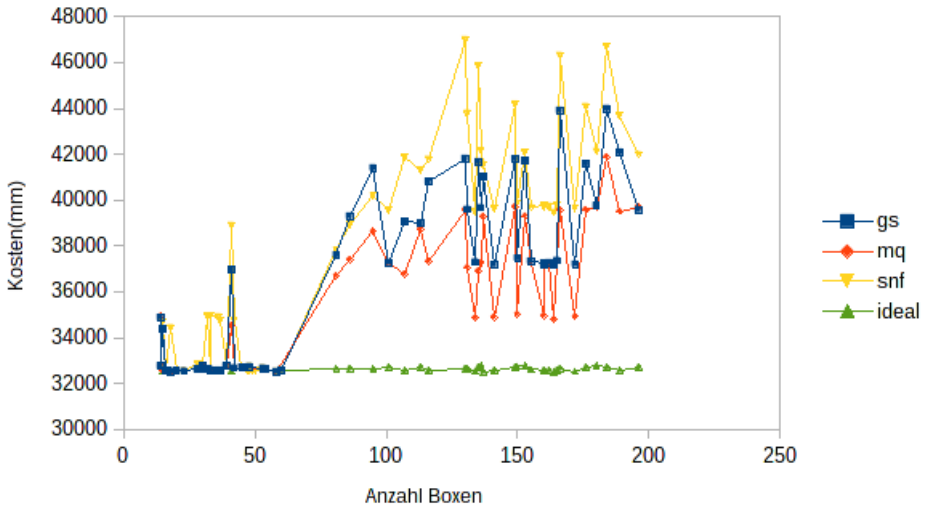

Abbildung 4.1: Gegenüberstellung der Güte der einzelnen Algorithmen

lich. Daher wird für den Guillotinen-Schnitt nach dem Platzieren einer Box eine Variante gewählt, bei welcher der Bereich überhalb einer Box nicht zerteilt wird. So sollen möglichst lange freie Quader mit einer größeren Grundfläche erhalten bleiben. Dies entspricht dem mittleren Ast in Abbildung 3.8. Die Implementierung der Algorithmen wurde in C++ (Standard C++11) realisiert und auf einem System mit folgenden Ressourcen getestet:

- **CPU:** Intel®Core™i5-4210M CPU @ 2.60GHz
- **Hauptspeicher:** 12GB
- **Betriebssystem:** Arch Linux 64Bit, Kernel: 4.4.3-1-ARCH.

Um das Abkühlen zu simulieren wurden die Werte von Vaeth aus [11] übernommen: $T = 50$, $T_{min} = 0.01$, $\sigma_N = 750$, $\tau = 0.95$.

In Abbildung 4.1 werden die einzelen Algorithmen bezüglich ihrer erreichten Kosten verglichen. Sehr deutlich zeigt sich dabei, dass das Maximal-Quader-Verfahren (mq) mit zunehmender Anzahl zu platzierender Boxen bessere Ergebnisse erzielt als das Guillotinen-Schnitt-Verfahren (gs) und weitaus bessere Ergebnisse als das Shelf-Next-Fit-Verfahren (snf).

In Tabelle 4.1 werden die Kosten und Rechenzeiten noch einmal detailliert aufgeschlüsselt. Ein Blick auf diese zeigt, dass die Berechnungszeiten des Maximal-Quader-Algorithmus mit zunehmender Anzahl an Boxen um ein Vielfaches höher sind als bei den übrigen Verfahren. Dieses Ergebniss war hinsichtlich der Komplexität des Verfahrens zu erwarten. Auch die Laufzeiten der übrigen Verfahren bestätigen unsere vorangegangenen Überlegungen.

Tabelle 4.1: Aufstellung der erreichten Kosten und Rechenzeiten

Anzahl Boxen/Referenzwert	Verfahren	Kosten	Rechenzeit in Sekunden
30/32767	SNF	32710	0.006555
	GS	32710	0.097688
	MQ	32710	0.458678
33/32767	SNF	32574	0.006918
	GS	32574	0.113193
	MQ	32574	0.558247
17/32764	SNF	0	0.00478
	GS	32577	0.042491
	MQ	0	0.092284
33/32765	SNF	34848	0.007045
	GS	32532	0.122661
	MQ	32532	0.459115
36/32764	SNF	34965	0.007158
	GS	32645	0.128514
	MQ	32645	1.45704
20/32765	SNF	32607	0.005138
	GS	32607	0.046004
	MQ	32607	0.139975
42/32767	SNF	34618	0.008341
	GS	32572	0.17819
	MQ	32572	1.26367

weiter auf nächster Seite

Tabelle 4.1 – *Fortsetzung*

Anzahl Boxen/Referenzwert	Verfahren	Kosten	Rechenzeit in Sekunden
39/32765	SNF	32643	0.00782
	GS	32643	0.135216
	MQ	32643	1.12988
23/32765	SNF	32646	0.005394
	GS	32646	0.05781
	MQ	32646	0.089788
45/32766	SNF	32525	0.008643
	GS	32525	0.168817
	MQ	32525	1.36327
30/32764	SNF	32554	0.0066
	GS	32554	0.096636
	MQ	32554	0.492863
28/32765	SNF	32802	0.006331
	GS	32581	0.114283
	MQ	32581	0.481074
14/32764	SNF	34764	0.004274
	GS	34764	0.033527
	MQ	34884	0.093011
41/32764	SNF	39068	0.008295
	GS	37136	0.157806
	MQ	34703	3.47986
16/32764	SNF	32754	0.004409
	GS	32754	0.052651
	MQ	32754	0.317643
14/32766	SNF	32579	0.004285
	GS	32579	0.048642
	MQ	32579	0.242314

weiter auf nächster Seite

Tabelle 4.1 – *Fortsetzung*

Anzahl Boxen/Referenzwert	Verfahren	Kosten	Rechenzeit in Sekunden
37/32767	SNF	34687	0.007739
	GS	32623	0.205199
	MQ	32623	4.26766
32/32764	SNF	34961	0.007004
	GS	32671	0.11833
	MQ	32671	1.7582
18/32767	SNF	34629	0.004763
	GS	32727	0.059928
	MQ	32727	0.377961
15/32765	SNF	34350	0.004595
	GS	34484	0.033486
	MQ	32646	0.213657
48/32766	SNF	32700	0.009085
	GS	32700	0.168638
	MQ	0	1.82143
60/32767	SNF	32579	0.010894
	GS	32579	0.222375
	MQ	0	3.79055
50/32766	SNF	32688	0.009296
	GS	0	0.182202
	MQ	0	2.6111
48/32766	SNF	32513	0.00901
	GS	0	0.18256
	MQ	0	2.1344
58/32765	SNF	32736	0.01066
	GS	32736	0.217047
	MQ	32736	3.75411

weiter auf nächster Seite

Tabelle 4.1 – *Fortsetzung*

Anzahl Boxen/Referenzwert	Verfahren	Kosten	Rechenzeit in Sekunden
46/32766	SNF	32703	0.008683
	GS	32703	0.219206
	MQ	0	3.84256
49/32764	SNF	32707	0.00928
	GS	0	0.175454
	MQ	0	2.26045
54/32767	SNF	32550	0.010367
	GS	32550	0.215663
	MQ	0	3.13603
53/32765	SNF	32509	0.009722
	GS	32509	0.200565
	MQ	32509	2.08363
47/32767	SNF	32560	0.008812
	GS	0	0.194151
	MQ	0	2.53932
101/32767	SNF	39469	0.017013
	GS	37207	0.327637
	MQ	37188	4.92171
116/32766	SNF	41875	0.019177
	GS	39373	0.441949
	MQ	37423	6.97497
86/32765	SNF	38961	0.014571
	GS	39347	0.282683
	MQ	37446	3.14525
131/32764	SNF	43765	0.021538
	GS	39575	0.64743
	MQ	37055	15.9629

weiter auf nächster Seite

Tabelle 4.1 – *Fortsetzung*

Anzahl Boxen/Referenzwert	Verfahren	Kosten	Rechenzeit in Sekunden
107/32765	SNF	42080	0.017712
	GS	39280	0.43428
	MQ	37001	6.07937
81/32765	SNF	37797	0.014067
	GS	37591	0.229505
	MQ	36688	2.17528
136/32765	SNF	41891	0.023564
	GS	39429	0.605943
	MQ	37029	9.19503
135/32767	SNF	45756	0.021907
	GS	41099	0.529563
	MQ	37015	12.7738
95/32766	SNF	40139	0.016001
	GS	41376	0.300321
	MQ	37979	2.75253
113/32767	SNF	41131	0.018581
	GS	38808	0.413589
	MQ	38556	6.74847
153/32764	SNF	41991	0.024519
	GS	41661	0.520378
	MQ	39252	6.50939
176/32764	SNF	43880	0.027839
	GS	41400	0.615071
	MQ	39399	10.1958
137/32766	SNF	41539	0.022515
	GS	41004	0.5144
	MQ	39292	6.20389

weiter auf nächster Seite

Tabelle 4.1 – *Fortsetzung*

Anzahl Boxen/Referenzwert	Verfahren	Kosten	Rechenzeit in Sekunden
166/32764	SNF	46208	0.026891
	GS	43870	0.670535
	MQ	39518	6.43526
180/32765	SNF	41899	0.028429
	GS	39551	0.631478
	MQ	39486	12.951
130/32765	SNF	46970	0.021423
	GS	41791	0.404071
	MQ	39563	5.05112
189/32766	SNF	43766	0.029627
	GS	42200	0.747823
	MQ	39625	17.864
184/32767	SNF	46710	0.029099
	GS	44011	0.730418
	MQ	39725	13.8186
149/32765	SNF	44142	0.024367
	GS	41759	0.564363
	MQ	39672	7.57555
196/32764	SNF	41844	0.030678
	GS	39442	0.712651
	MQ	39599	13.022
141/32766	SNF	39618	0.022519
	GS	37173	0.501
	MQ	34881	6.8476
164/32765	SNF	39676	0.026076
	GS	37367	0.665341
	MQ	35020	12.6976

weiter auf nächster Seite

Tabelle 4.1 – *Fortsetzung*

Anzahl Boxen/Referenzwert	Verfahren	Kosten	Rechenzeit in Sekunden
160/32767	SNF	39903	0.025487
	GS	37419	0.629974
	MQ	35120	10.0463
162/32767	SNF	39759	0.025518
	GS	37338	0.758195
	MQ	37302	16.5028
165/32764	SNF	39800	0.026125
	GS	37417	0.713387
	MQ	37354	17.3068
134/32765	SNF	39495	0.021634
	GS	37284	0.501426
	MQ	34859	7.94989
155/32765	SNF	39636	0.024563
	GS	37302	0.631728
	MQ	37235	9.50017
172/32766	SNF	39760	0.027181
	GS	37322	0.773039
	MQ	35039	16.7013
150/32764	SNF	39822	0.023754
	GS	37474	0.554787
	MQ	35044	10.8941
160/32764	SNF	39899	0.025284
	GS	37383	0.608919
	MQ	37452	15.8487

5 Fazit

Ziel dieser Arbeit war die Entwicklung einer Heuristik für das dreidimensionale Bin-Packing-Problem unter zusätzlichen Restriktionen. Dazu wurden zunächst verschiedene Konstruktionsalgorithmen an das Problem angepasst und implementiert. Diese laden jeweils eine Sequenz von Boxen nacheinander in einen bzw. mehrere Container (Bins). Um die Güte der so produzierten Lösungen zu verbessern verwendeten wir das sogennante simulierte Abkühlen. Da die Konstruktionsalgorithmen alle deterministisch arbeiten, konnten aus einer Lösungen durch Tausch einzelner Boxen der Beladesequenz oder ganzer Gruppen leicht weitere Lösungen erzeugt werden. Darüber hinaus konnte so ein verharren in einem lokalen Maximum vermieden werden. Da viele Boxen gleiche Abmessungen aufwiesen, wurden im Anschluß Boxen gleicher Abmessung vertauscht sofern die schwerere Box oberhalb der leichteren platziert war. Im Effizienzvergleich haben wir gesehen, dass der einfachste Konstruktionsalgorithmus, der Shelf-Next-Fit-Algorithmus, die beste Rechenzeit erzielt. Gefolgt wird dieser von dem Guillotinen-Schnitt-Verfahren und das Schlusslicht bildet der Maximal-Quader-Algorithmus. Allerdings ist auch bei diesem, für die von uns betrachteten Instanzen, der Rechenaufwand durchaus vertretbar. Weiter hat sich gezeigt dass die zusätzliche Rechenzeit zu einer weitaus besseren Ausnutzung des Containervolumens führt. So erzielte hier stets das Maximal-Quader-Verfahren die besten Ergebnisse.

Nun gibt es noch einige Möglichkeit um die Heuristik noch weiter zu optimieren, welche aber durch den begrenzten zeitlichen Rahmen für dieses Projekt nicht mehr umgesetzt werden konnten. Dazu sei zunächst auf eine Methode von Zhang et al. [12] verwiesen. Dabei werden alle bereits platzierten Boxen mit der zu platzierenden Box verglichen. Ist die aktuelle Box größer als die be-

reits gesetzte, so werden diese gegeneinander ausgetauscht. Anderfalls wird versucht die an die platzierte Box angrenzenden Boxen soweit zu verschieben, dass die größere Box eingefügt werden.

Ein weitere Optimierungsmöglichkeit entsteht aus dem Ansatz einen Container nicht zu schließen, falls die zu platzierende Box nicht mehr in diesem verstaut werden kann, sondern erst zu prüfen, ob sich unter den folgenden Boxen der aktuellen Beladesequenz eine Box gleicher Gruppe befindet, welche noch in den Container passt, und diese gegen die aktuelle Box zu tauschen. Dieser Ansatz liesse sich darüber hinaus mit der Methode von Zhang et al. kombinieren.

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] S. Ceschia and A. Schaerf. *Local Search for a Multi-Drop Multi-Container Loading Problem*. Journal of Heuristics, 2013.

[2] J. Egeblad and David Pisinger. *Heuristic approaches for the two- and three-dimensional knapsack packing problems*. 2006.

[3] E. Falkenauer, A. Delchambre, and E. Falkenauer A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1186–1192, 1992.

[4] Michael R. Garey and David S. Johnson. *A 71/60 theorem for bin packing*1*. Journal of Complexity, 1985.

[5] Jukka Jylänki. *A Thousand Ways to Pack the Bin - Practical Approach to Two-Dimensional Rectangle Bin Packing*. 2010.

[6] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[7] D. Mack and A. Bortfeldt. *Eine Heuristik für das mehrdimensionale Bin-Packing-Problem*. Diskussionsbeiträge Fachbereich Wirtschaftswissenschaft. Fernuniversität Hagen, Fachbereich Wirtschaftswiss., 2008.

[8] Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000.

[9] Guntram Scheithauer. *A Three-dimensional Bin Packing Algorithm*. J. Inform. Process. Cybernet. EIK 27, 1991.

[10] Johannes Terno, Guntram Scheithauer, Uta Sommerweiß, and Jan Riehme. An efficient approach for the multi-pallet loading problem. *European Journal of Operational Research*, 123(2):372 – 381, 2000.

[11] T. Väth. *Entwicklung einer anwendungsorientierten Packheuristik mit Hohlraumfragmentierung*. Karlsruhe Institute of Technology, 2014.

[12] Zhaoyi Zhang, Songshan Guo, Wenbin Zhu, Wee-Chong Oon, and Andrew Lim. *Space Defragmentation Heuristic for 2D and 3D Bin Packing Problems*. Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, 2011.

Danksagung

Danken möchte ich als erstes meinem Betreuer, Herrn Prof. Dr. Peter Tittmann, für seine hervorragende Lehre, welche die Grundlagen zur Erstellung dieser Arbeit bereitete, für seine Unterstützung und Betreuung.

Danken möchte ich auch dem Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme und besonders meiner Betreuerin, Frau Dipl.-Math. (FH) Franziska Heinicke, welche das Thema dieser Arbeit stellte und mich stets mit konstruktiver Kritik und kritischem Hinterfragen der Fertigstellung dieser Bachelorarbeit näher brachte. Vielen Dank für Zeit und Mühen, die Du in meine Arbeit investiert hast.